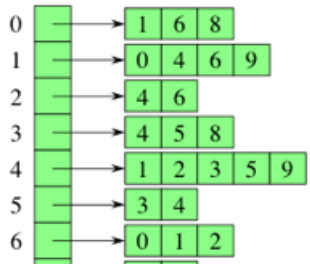
פירוט המטלה

המימוש התבצע בשפת #C ע"י מבני נתונים בשם "List" או בשמו המלא "Adjacency Lists"(רשימת סמיכויות/סמיכות) , שקול ל- Vector ב ++C ע"י שימוש בספרית STL. המטלה רצה 4 שעות בלבד עבור 1000 צמתים, ו500 הרצות גרפים אקראיים, במחשב נייח בן 8 שנים, לכן הסיבוכיות טובה מאוד ויעילה.

דוגמא להמחשה:



ייצוג גרף עם רשימות סמיכות משלב מטריצות סמיכות עם רשימות של צלעות. עבור כל צומת ,מאחסן מערך של צמתים הסמוכים אליו. בדרך כלל יש לנו מערך של רשימת סמיכות אחת לכל צומת.
מספרי הצמתים ברשימת סמיכות אינם נדרשים להופיע בשום סדר מסוים, אם כי לעיתים קרובות נוח לרשום אותם בסדר הולך וגדל, כמו בדוגמא מעלה.

אנו יכולים להגיע לרשימת הסמיכות של כל צומת **<u>בזמן קבוע</u>** , מכיוון שעלינו רק להוסיף לאינדקס למערך. כדי לברר אם הצלע(i,j) קיים בגרף, נעבור לרשימת הסמכות של i בזמן קבוע ואז נחפש j בסמיכות של i ברשימה. כמה זמן לוקח במקרה הגרוע ביותר? התשובה היא $\Theta(d)$.
כאשר d הוא דרגת צומת של i , כי זה כמה רשימת הסמכות של i ארוכה.
הדרגה של הצומת i יכולה להיות גבוהה כמו |V|-1| או נמוך מאוד כמו 0(אם אין צלעות בכלל).
בגרף לא מכוון, צומת j נמצאת ברשימת הסמיכות של צומת i אם ורק אם i נמצא ברשימת הסמיכות של j .
כמה מקום תופסות רשימות סמיכות? יש לנו |V| רשימות, וכל רשימה יכולה להכיל עד |V|-1| צמתים, סה"כ רשימת סמכויות לגרף לא מכוון מכילה 2|E| אלמנטים.

למה **2|E|** צלעות? כל צלע (i,j) מופיעה בדיוק פעמיים ברשימת סמיכויות, פעם אחת ברשימת i , ופעם אחת ברשימת j , ויש שם |E| צלעות.

סיבוכיות:

For finding the neighbors of vertex v:
Edge List: O(|E|)
If the list is unsorted you need to check every edge to see if it comes from v
For a complete graph (where every vertex is connected to all other vertices) this would be O(|V|^2)

Adjacency Matrix: O(|V|)
You need to check the the row for v, (which has |V| columns) to find which ones are neighbors

Adjacency List: O(|N|)(רשימת סמיכות)
where N is the number of neighbors of v
we need to iterate through the list for the vertex v, which has N neighbors
v could have up to V-1 neighbors, so we could also say it is O(|V|)

The adjacency list contains:
-1 array per vertex, for a total of V arrays (we are only considering the space for the array pointer, not the contents of the array).
-Each directed edge is contained once somewhere in the adjacency list, for a total of E edges (a bidirectional edge is just 2 directed edges, so if we are using bidirectional edges it would just be 2E).

V+E is Θ(V + E)

מה הסיבה שהשתמשנו בסוג של Adjacency List(רשימת סמיכות)?

הסיבות הן רבות, אך נפרט בקצרה:
רצינו בעיקר גמישות בגודל, וגישה נוחה בסוג מבני הנתונים הזה, וניתן לאחסן בו כמה נתונים שאנחנו צריכים),ובנוסף מבנה הנתונים הזה יעיל מאוד והסיבוכיות טובה מאוד, כפי שמפורט מעלה.

תיאור הפונקציות בקוד

1. Build_random_graph – יצרנו מחלקה בשם Graph שמקבלת בנאי את הצומת ואת ההסתברות ליצירת צלע.
כל עיניין הרנדומליות (ההסתברות) התבצעו ע"י הגדרה שהחלטנו ש- 1000 יהיה הקבוע שלנו(Const) דוגמא: אם בrand יצא הסתברות 0.2 אז בעצם 200 מתוכם יקיימו את התנאי שאכן קיים צלע, ו – 0.8 לא קיימו את התנאי של צלע(ז"א שלא קיימת צלע).
ע"י GraphList מוסיפים לNode עם צומת ,ורצים בלולאה וסופרים כמה צמתים וכמה צלעות יש שיש להם שכנים.
לבסוף מדפיסים את הגרף על מטודה PrintGraphData שמדפיסה את כל הנתונים הרלוונטיים לגרף. (יש דוגמא בקוד לגרף לדוגמא של ההדפסה).
סיבוכיות $O(\frac{v^2}{2})= (v^2)$

2. Is_Isolted – בודקת אם לצמתים יש שכנים או אין שכנים במידה ויש צומת אחת ללא שכנים מחזירה 1 אחרת מחזירה 0.
מחזיק את התוצאה ב –answer ורץ על כל השכנים בגרף ולבסוף מחזיר את answer.
סיבוכיות – O(V)

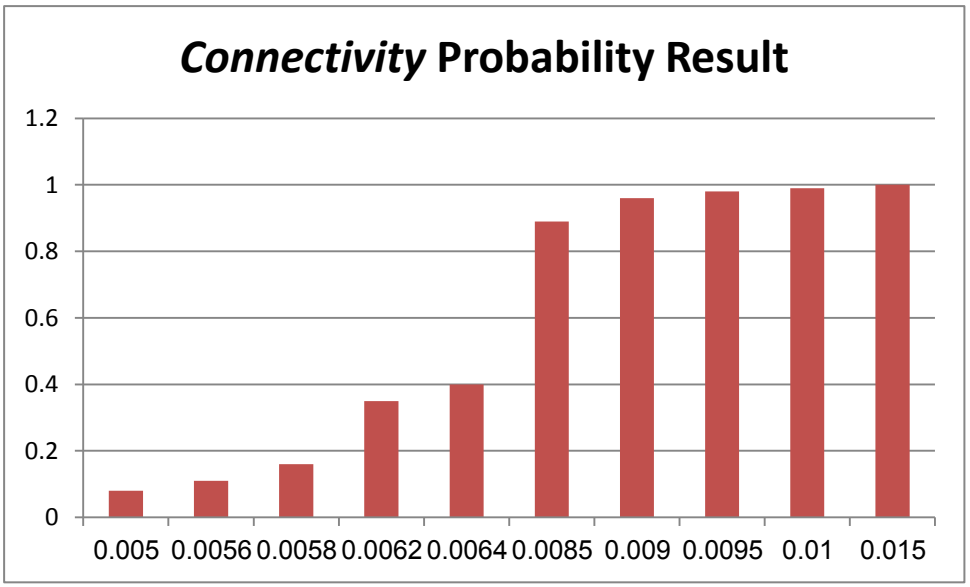
3. Connectivity – מתבסס על BFS בעצם ע"י תור(queue) מוסיף לתור את הצמתים ע"י אלגוריתם BFS שבודק את השכנים של כל אינדקס של צומת, וכאשר האינדקס של הצומת הסתיים מוחק Enqueue וממשיך הלאה באלגוריתם.
במידה והמערך visited מצא שכן הוא על true במידה ואין שכנים הוא על false וכך סופר צומת צומת.
לבסוף רץ בלולאה על המשתנה answer ובודק אם המערך שונה באינדקס במידה והגרף קשיר מחזיר 1 אם לא קשיר מחזיר 0 ע"י answer.
סיבוכיות – O(V+E),BFS

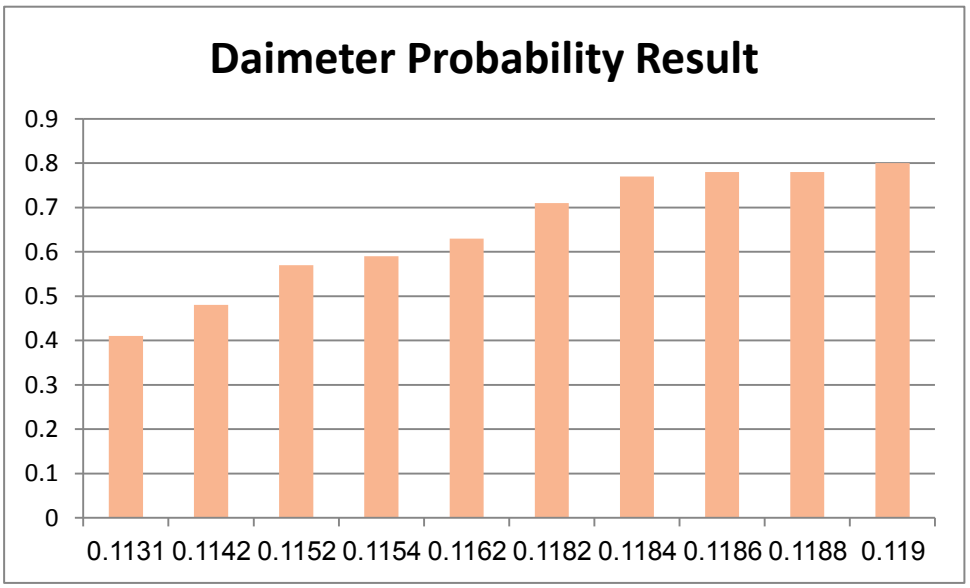
4. Diameter - משתמשת במטודה connectivity במידה וזה שונה מ – 1 הdiam הוא אינסוף(בקוד רשום 1-).
הdiameter משתמש גם במטודה BFS כדי למצוא את האורך של הdiameter את הגודל המקסימלי של diam במרחק מינימלי, ומחזירה לבסוף את המקסימום diam.
סיבוכיות – O(V*(V+E),BFS on each V

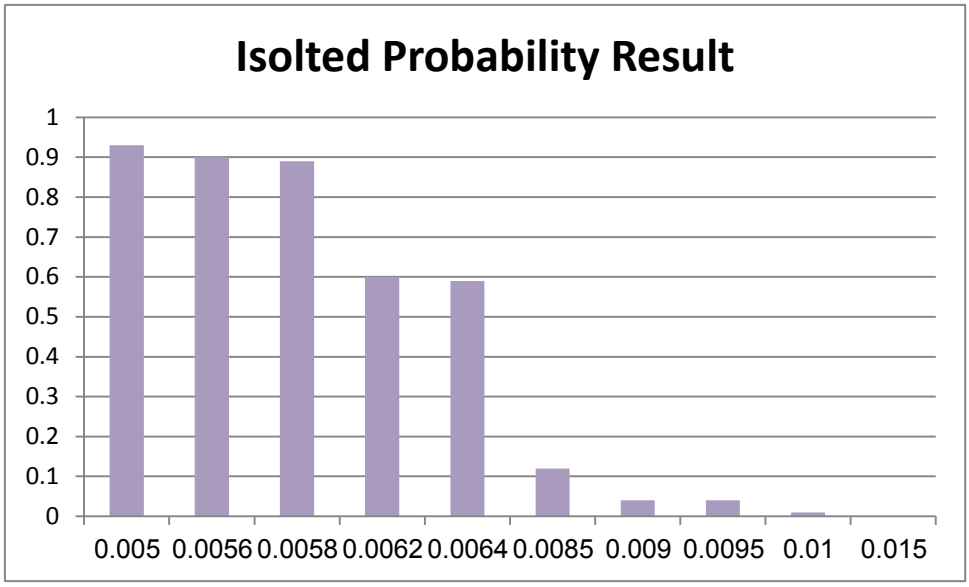
5. BFS – אלגוריתם ידוע שהוכח עוד בקורס אלגוריתמים 1. המימוש זהה.

המשמש למעבר על צומתי הגרף, תוך חיפוש צומת המקיים תכונה מסויימת.
צומת כלשהו בגרף נקבע להיות צומת ההתחלה V1, והאלגוריתם עובד על כל הצמתים במרחק של צלע אחת מ – V1 , ואז כל הצמתים במרחק 2 צלעות מ V1 וכן הלאה.
סיבוכיות – O(|V|+|E|)

גרפים באקסל – תוצאות הסימולציות







ניתן לראות שכל התכונות של הגרפים לפי המטלה אכן מתקיימות, לאחר חקירה ארוכה של הגרפים, התוצאות מדברות בעד עצמן, והתכונות של הגרפים האקראיים מתקיימות.

- לגבי גרף הקשירות - ניתן לראות כאשר ההסתברות קטנה יותר מ-threshold לפי הגרף רואים שהגרף לא קשיר בהסתברות גבוהה, ובמידה וההסתברות גדולה יותר מ-threshold לפי הגרף רואים שהגרף קשיר בהסתברות גבוהה. הגרף מונוטוני עולה ככל שההסתברות גדלה.

- לגבי גרף הקוטר - ניתן לראות כאשר ההסתברות גדולה יותר מ-threshold לפי הגרף רואים שבגרף הקוטר שווה ל-2 בהסתברות גבוהה ואכן התכונה מתקיימת לפי הגרף. הרבה יותר גרפים מקיימים את התכונה ככל שעולים בהסתברות.

- לגבי גרף צומת מבודד – ניתן לראות כאשר ההסתברות קטנה יותר מ-threshold לפי הגרף רואים שבגרף קיים צומת מבודד בהסתברות גבוהה, ובמידה וההסתברות גדולה יותר מ-threshold לפי הגרף רואים שהגרף לא קיים צומת מבודדת.

<u>לסיכום</u>

כל הטענות של המטלה אכן מתקיימות וניתן לראות זאת ע"י הגרפים שהוכנו.
המטלה רצה 4 שעות בלבד עבור 1000 צמתים, ו500 הרצות גרפים אקראיים, במחשב נייח בן 8 שנים, לכן הסיבוכיות טובה מאוד ויעילה.